PCT/IL2004/001066
27 JAN 2005

PA 1268128

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

January 05, 2005

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE UNDER 35 USC 111.

APPLICATION NUMBER: *60/532,947*
FILING DATE: *December 30, 2003*

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

[illegible]

E. BORNETT
Certifying Officer

BEST AVAILABLE COPY

13281 U.S. PTO

Please type a plus sign (+) inside this box → +

Docket Number:	1893/27

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53 (c).

00727 U.S. PTO 60/532947 123003

INVENTOR(S)/APPLICANT(S)

Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
LEEOR	AHARON	TEL AVIV, ISRAEL
CFIR	COHEN	TEL AVIV, ISRAEL

☐ Additional inventors are being named on page 2 attached hereto

TITLE OF THE INVENTION (280 characters max)

UNIVERSAL WORM CATCHER

CORRESPONDENCE ADDRESS

Direct all correspondence to:

☐ Customer Number → Place Customer Number Bar Code Label here

OR

☐ Firm or Individual Name	Mark M. Friedman				
Address	c/o DISCOVERY DISCOVERY				
Address	9003 FLORIN WAY				
City	UPPER MARLBORO	State	MD	ZIP	20772
Country	US	Telephone	301-952-1011	Fax	301-952-9023

ENCLOSED APPLICATION PARTS (check all that apply)

☒ Specification — Number of Pages: 6

☐ Drawing(s) — Number of Sheets:

☒ Other (specify): ASSIGNMENT

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT (check one)

☐ A check or money order is enclosed to cover the filing fees

☒ The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number: 06-2140

FILING FEE AMOUNT: $160 $40

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are:

Respectfully submitted,

SIGNATURE [illegible] DATE 24 DEC 03

TYPED or PRINTED NAME Mark M. Friedman REGISTRATION NO. (if appropriate) 33,883

TELEPHONE (703) 415-1581

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

SEND TO: Box Provisional Application, Assistant Commissioner for Patents, Washington, DC 20231

P19LARGE/REV04

Check Point
SOFTWARE TECHNOLOGIES LTD
We Secure the Internet.

Universal Worm Catcher Information for patent

DRAFT

CONFIDENTAL

<<Name>>
Products Department
14-Dec-03

CONNECT PROTECT MANAGE ACCELERATE

Target date for product launch:

The feature is planned for Feb-2004.

Introduction:

Many network attacks are based on sending malicious code (a.k.a Worms) in an unexpected way over network connections where no code is expected in advance. An application on the attacked (target) computer receiving the code is then tricked to execute the code through various known weaknesses.

Detection of such attacks is different from Anti-Virus scanning that is performed on a legitimate transfer of executable code which might contains a malicious virus. This is also different from a regular firewall behavior which may block any traffic that is marked to contain executable code.

Benefit of the patent:

1) Detect and block network traffic that contains malicious code.
2) The detection is generic; there is no need for a predefined signature for every new malicious code found.
3) Once an attack has been detected at the network, it can be blocked before ever reaching its destination.
4) Detection is done in a central location (e.g., firewall) and there is no need to install the program on every client/server.

Description:

1) The detection and blocking is performed at the network level, it can be performed on a perimeter gateway (e.g. firewall) on an internal network router or switch or it can be performed at the network level on the operating system of the client or server itself.
2) Network traffic is pre-filtered and only traffic that is highly likely to contain such an attack is further examined. Pre-filtering is performed according to network protocol being used (e.g. HTTP) it can filter only parts of the protocol that are known to contain an attack (e.g. HTTP Headers) and the filter may be applied only if there is data content that is not likely for such a protocol (e.g. non-ASCII characters.)
3) It is assumed that the machine-language-instructions of the target of the attack is known (e.g. Intel's Pentium.) The suspected data is disassembled and a decision based on an accumulated "threat-weight" is performed.
3A) The start location of disassembly is started according to:
3Ai) Enumerating over all possible locations in the suspected data.
3Aii) Identifying special sequence of instructions that are usually used to initiate such exploits.
3B) The "threat-weight" is increased whenever:
3Bi) Legal machine language instruction is encountered.

3Bii) Decreased whenever an illegal instruction is encountered.
3Biii) The "threat-weight" is further increased whenever an instruction that is more likely to be in an attack is detected (e.g. system calls and instructions that are less sensitive to their location in memory.)
4) A detection is made when the "threat-weight" pass a fixed threshold level. Once detection was made, an alert can be generated, the traffic can be blocked, and it is possible to subsequently block all traffic from the source of the attack.

Prior Art:

1) A similar patent already exist US patent 6,301,699 and an implementation of this patent currently exist. However:
1A) It is focused on a specific weakness of application servers called "buffer overflow".
1B) It looks as if the solution of the patent is a software that is installed as an add-on to the application server itself and it captures suspected data at the locations in which functions in the code of the application server are invoked. In our solution the data is captured at the network level, possible in a separate computer/device.
1C) It is not completely clear from the patent how their invention works and if it is identically to the description given above.

Technical Details:

Locating the start of a Buffer Overflow attack on Windows target

As said before there are several mechanisms by which an attacker can cause a code to be executed on a targeted machine. One such mechanism is using a weakness of an application on the target machine which is called "buffer overflow" or "boundary condition error". In essence, in a buffer overflow attack the attacker crafts a network message that is received by a specific application running on the target machine. The application copies the buffer into an internal buffer that is too small to contain it and by this causing a memory corruption which leads to arbitrary code execution. An attacker can then tune the message content such that code that was carried in the message itself will be executed by the target.

Usually both code and data of the application are stored in the same memory. The application code is made from functions and each "parent" function can call the execution of a "child" function. Before passing execution to a child function, the parent should keep a record of where execution should continue in the parent code once the child returns from its execution. The return address is placed (or pushed) by the parent in a special memory area called the "stack".

Now let us focus on such a child function in which an attack is taking place, it is common for such a function to copy an incoming message to a buffer that is also located in the stack next to the return address. If the message is too big and if the function code is not carefully written then the content of the message will overwrite the return address and upon completion, the child function will return execution to a new memory location

which is determined by the content of the message. The goal of an attacker is to craft a message that will set the new execution to the stack area containing other parts of the copied message. On Unix platforms (Linux/Solaris) the exact location of the stack in memory is known in advance to the attacker and therefore he can know in advance what return address should be used. Every application may have a weakness in a different function and therefore every new worm will have a different return address and therefore it is not possible to easily identify all such attacks using a single generic filter. In contrast on Windows platforms the location of the stack is not fixed and the attacker is forced to use a special code that will ensure that the return address will fall on the code that he supplied in the message. This special code is generic to most (about 80%) of these attacks on Windows and therefore it is easy to detect them generically (universally.)
More specifically modules of the operating system (e.g., kernel32.dll) that are used by the application are loaded into known memory locations. The attacker modify the return address to a location in one of these modules that happens to contain a machine code that return execution to the stack where the rest of the message is located (on Intel's Pentium this machine code is called "CALL ESP" and its operation code is FF D4.) A generic detector for Windows can search for "vulnerable return address" in the message that happens to fall in the memory area where operating system modules are located.

Not every detection of a "vulnerable return address" is necessarily a worm, to complete the detection it is needed to disassemble the code which immediately follows the "vulnerable return address".

Disassembling a suspected code

Disassembling a suspected message can be the final verdict to the existence of an attack. An "ANALYZER" program can be written to perform an automatic disassembly on every suspicious message. However, in order to perform a disassembly the ANALYZER needs to decide where the code starts, on Windows in some cases this can be detected through a detection of a "vulnerable return address" as described above but in some Windows attacks and in all Unix attacks such a detection does not exist and the ANALYZER should try to start and disassemble the code from every offset within the suspected message. It is therefore necessary to limit the search only to suspicious parts of messages as described before.

Each different disassemble task is performed by a sub-process called a "SPIDER". Each SPIDER creates a "FLOW" structure which follows the instructions in the code one by one. Every time a conditional jump instruction is meet the FLOW is split into two FLOWs each continuing in a different execution path, and as a result a single SPIDER may contain a list (or tree) of FLOWs.

Each FLOW maintains a "threat weight" and as the FLOW progresses along the code its threat-weight is updated:

- Invalid instruction (non-code) will decrease the threat weight by a given amount. For example, for Windows running on Intel's Pentium:
    - Illegal instructions: "IN 64", "ARPL".
    - Uncommon instructions: "DAA", "SHAF".

    - Invalid memory access: "MOV [00000044], EAX" or "ADD [EBX+12345678], ECX" (when EBX is not initialized)
- Valid instruction increases the weight.
- There are instructions (or set of instructions a.k.a. Meta-instruction) that are likely to appear in the initialization code of a worm and therefore their existence will increase the weight even more. For example on Windows:
    - The attacker does not know in advance what will be the absolute address in which his code will be executed. In order to know it, he can load the address (EIP) in runtime into a register (EBP) using the following sequence (the known "call delta" technique)
        - 1: CALL 2 // Perform a function call to the code that starts at an offset of 2 bytes, which "happens" to be the next instruction. This will push the current address in stack
        - 2: POP EBP // Pop the stack into the EBP register.
    - Knowing the code location can also be computed from the stack location which can be read through a command like:
        - MOV EBP, ESP
    - A worm will want to perform system calls in order to perform "useful" actions (e.g. damage or propagation) in order to perform such system calls it is useful to know where the Kernel32.dll is located. For this the following operation is usually performed:
        - MOV EAX, FS:[30]
    - In some cases the worm code is scrambled in order to hide its existence, however in this case there must be a descrambling code at the start which has a fixed sequence that can be easily detected.

A SPIDERS's threat weight is equal to the maximum threat weight of its FLOWs. Different FLOWs in the same SPIDER can communicate with each other. In case one FLOW encounters an invalid-instruction, it can "tell" the other FLOWs in this SPIDER about the event, and they, in turn, will decrease their threat-weight.

Future plans:

- Keep a list of exact vulnerable address in Windows
- Full CPU simulation
- Full memory (stack status) simulation

We claim:

1. One or more aspect of universal worm cathcer substantially as described herein.

Document made available under the Patent Cooperation Treaty (PCT)

International application number:	PCT/IL04/001066
International filing date:	18 November 2004 (18.11.2004)

Document type:	Certified copy of priority document	
Document details:	Country/Office:	US
	Number:	60/532,947
	Filing date:	30 December 2003 (30.12.2003)

Date of receipt at the International Bureau: 08 February 2005 (08.02.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)